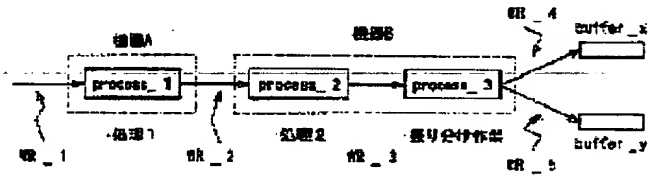
d)

DEVICE AND METHOD FOR PROGRAM GENERATION SUPPORT

Patent number: JP9050371
Publication date: 1997-02-18
Inventor: TERAO SO; YOSHIKAWA NORIO; MORI KENICHIRO
Applicant: OMRON TATEISI ELECTRONICS CO
Classification:
- international: ***G06F9/06; G06F17/00;*** **G06F9/06; G06F17/00;** (IPC1-7): G06F9/06; G06F17/00; G06F17/60
- european:
Application number: JP19950200954 19950807
Priority number(s): JP19950200954 19950807

Abstract of **JP9050371**

PROBLEM TO BE SOLVED: To use the control structure of a model prepared in a virtual environment by a simulator as the upstream design tool of actual system control by structuring a system according to a concept able to be shared with actual systems. SOLUTION: Processes (process 1, process 2, process 3) provided with a data processing content to be executed by equipment in the actual system as processing programs on data in itself and to manage their execution are set from a simulation model from the simulator, and Work routes (WR 1, WR 2, WR 3) are set by designating an order to make Work transit between set processes, and the program of the simulation model is prepared based on the operation setting of set processes and Work routes, and also, an execution program is prepared by linking them.



(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号

特開平9－50371

(43)公開日　平成９年(1997)２月18日

(51)Int.Cl.[6]		識別記号	庁内整理番号	ＦＩ		技術表示箇所
Ｇ０６Ｆ	9/06	５３０		Ｇ０６Ｆ	9/06	５３０Ｔ
						５３０Ｈ
	17/00		7925－5Ｌ		15/20	Ｄ
	17/60				15/21	Ｒ

審査請求　未請求　請求項の数18　ＯＬ　（全 13 頁）

(21)出願番号　特願平7－200954

(22)出願日　平成７年(1995)８月７日

(71)出願人　000002945
オムロン株式会社
京都府京都市右京区花園土堂町10番地

(72)発明者　寺尾　創
京都府京都市右京区花園土堂町10番地　オムロン株式会社内

(72)発明者　吉川　典雄
京都府京都市右京区花園土堂町10番地　オムロン株式会社内

(72)発明者　森　健一郎
京都府京都市右京区花園土堂町10番地　オムロン株式会社内

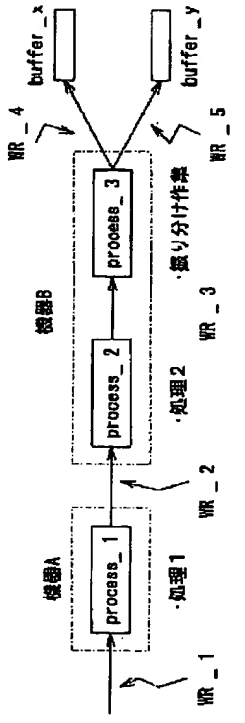
(74)代理人　弁理士　和田　成則

(54)【発明の名称】　プログラム作成支援装置およびプログラム作成支援方法

(57)【要約】

【課題】　実システムとの間で共有できる概念に従ってシステムの構造化を行うことにより、シュミレータでの仮想環境上で作成したモデルの制御構造を実システム制御の上流設計ツールとして利用する。

【解決手段】　シミュレータからのシミュレーションモデルから、実システム中の機器において実行すべきデータ処理内容を自身内のデータ上に処理プログラムとして有しその実行を管理するプロセス（ｐｒｏｃｅｓｓ＿１、ｐｒｏｃｅｓｓ＿２、ｐｒｏｃｅｓｓ＿３）を設定し、この設定されたプロセス間をＷｏｒｋが遷移していく順序を指定してＷｏｒｋ経路（ＷＲ＿１、ＷＲ＿２、ＷＲ＿３、ＷＲ＿４）を設定し、設定されたプロセスおよびＷｏｒｋ経路の動作設定に基づき、上記シミュレーションモデルのプログラムを作成するとともに、リンクして実行プログラムを作成する。



【特許請求の範囲】
【請求項１】　シミュレータからシミュレーションモデルを受け、実システムを稼働させる実行コードを作成するプログラム作成支援装置であって、
上記シミュレーションモデルから、上記実システム中の機器において実行すべきデータ処理内容を自身内のデータ上に処理プログラムとして有しその実行を管理するプロセスと、このプロセス間をＷｏｒｋが遷移していく順序を指定するＷｏｒｋ経路とからなるコントロールフローを記述したコントロールフロー記述プログラムを作成するコントロールフロー記述プログラム作成手段と、
このコントロールフロー記述プログラム作成手段で記述されたコントロールフロー記述プログラムに、上記プロセスが処理する処理プログラムを設定して上記実システムを稼働させる実行コードを作成する実行コード作成手段とを具備することを特徴とするプログラム作成支援装置。
【請求項２】　上記コントロールフロー記述プログラム作成手段は、
上記シミュレータからのシミュレーションモデルから、上記実システム中の機器において実行すべきデータ処理内容を自身内のデータ上に処理プログラムとして有しその実行を管理するプロセスを設定するプロセス設定手段と、
このプロセス設定手段で設定されたプロセス間をＷｏｒｋが遷移していく順序を指定するＷｏｒｋ経路を設定するＷｏｒｋ経路設定手段とを有することを特徴とする請求項１記載のプログラム作成支援装置。
【請求項３】　上記実行コード作成手段は、上記コントロールフロー記述プログラムに、ライブラリに格納された上記プロセスが処理する処理プログラムをインクルードして上記実システムを稼働させる実システム稼働プログラムを作成し、これをコンパイルし、さらにリンクして上記実行コードを作成することを特徴とする請求項１記載のプログラム作成支援装置。
【請求項４】　上記実行コード作成手段は、上記コントロールフロー記述プログラムに、エディタで記述された上記プロセスが処理する処理プログラムをインクルードして上記実システムを稼働させる実システム稼働プログラムを作成し、これをコンパイルし、さらにリンクして上記実行コードを作成することを特徴とする請求項１記載のプログラム作成支援装置。
【請求項５】　上記実行コード作成手段は、上記コントロールフロー記述プログラムをコンパイルし、これに上記プロセスが処理する処理プログラムをリンクして上記実行コードを作成することを特徴とする請求項１記載のプログラム作成支援装置。
【請求項６】　上記プロセス設定手段は、上記シミュレータと上記実システム間で共有する論理的工程をプロセスとすることを特徴とする請求項２記載のプログラム作成支援装置。
【請求項７】　上記プロセス設定手段は、オブジェト指向言語を使用して上記プロセスをオブジェトとして形成することを特徴とする請求項２記載のプログラム作成支援装置。
【請求項８】　Ｗｏｒｋ経路設定手段は、上記オブジェトのプロセス機能を実行可能なプログラム言語で記述する際に、中間言語を介してパラメータを入力するパラメータ入力手段を有することを特徴とする請求項２記載のプログラム作成支援装置。
【請求項９】　上記中間言語は、上記オブジェト指向言語で記述されることを特徴とする請求項８記載のプログラム作成支援装置。
【請求項１０】　シミュレータからシミュレーションモデルを受け、実システムを稼働させる実行コードを作成するプログラム作成支援方法であって、
上記シミュレーションモデルから、上記実システム中の機器において実行すべきデータ処理内容を自身内のデータ上に処理プログラムとして有しその実行を管理するプロセスと、このプロセス間をＷｏｒｋが遷移していく順序を指定するＷｏｒｋ経路とからなるコントロールフローを記述したコントロールフロー記述プログラムを作成し、
記述されたコントロールフロー記述プログラムに、上記プロセスが処理する処理プログラムを設定して上記実システムを稼働させる実行コードを作成することを特徴とするプログラム作成支援方法。
【請求項１１】　上記シミュレータからのシミュレーションモデルから、上記実システム中の機器において実行すべきデータ処理内容を自身内のデータ上に処理プログラムとして有しその実行を管理するプロセスを設定し、このプロセス設定手段で設定されたプロセス間をＷｏｒｋが遷移していく順序を指定するＷｏｒｋ経路を設定することを特徴とする請求項１０記載のプログラム作成支援方法。
【請求項１２】　上記コントロールフロー記述プログラムに、ライブラリに格納された上記プロセスが処理する処理プログラムをインクルードして上記実システムを稼働させる実システム稼働プログラムを作成し、これをコンパイルし、さらにリンクして上記実行コードを作成することを特徴とする請求項１０記載のプログラム作成支援方法。
【請求項１３】　上記コントロールフロー記述プログラムに、エディタで記述された上記プロセスが処理する処理プログラムをインクルードして上記実システムを稼働させる実システム稼働プログラムを作成し、これをコンパイルし、さらにリンクして上記実行コードを作成することを特徴とする請求項１０記載のプログラム作成支援方法。
【請求項１４】　上記コントロールフロー記述プログラ

ムをコンパイルし、これに上記プロセスが処理する処理プログラムをリンクして上記実行コードを作成することを特徴とする請求項１０記載のプログラム作成支援方法。

【請求項１５】　上記シミュレータと上記実システム間で共有する論理的工程をプロセスとすることを特徴とする請求項１１記載のプログラム作成支援方法。

【請求項１６】　オブジェト指向言語を使用して上記プロセスをオブジェトとして形成することを特徴とする請求項１１記載のプログラム作成支援方法。

【請求項１７】　上記オブジェトのプロセス機能を実行可能なプログラム言語で記述する際に、中間言語を介してパラメータを入力するパラメータ入力手段を有することを特徴とする請求項１１記載のプログラム作成支援方法。

【請求項１８】　上記中間言語は、上記オブジェト指向言語で記述されることを特徴とする請求項１７記載のプログラム作成支援方法。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、シミュレータからシミュレーションモデルを受け、実システムを稼働させる実行コードを作成するプログラム作成支援装置およびプログラム作成支援方法に関する。

【０００２】

【従来の技術】従来、離散系シミュレータ装置（例えば、米国Deneb Robotics社製のＱＵＥＳＴ等）は、最適なシステム構成の検討および稼働予想を事前に行うものである。この従来の離散系シミュレータ装置は、その利用形態として、実在するシステムの解析に限られ、機器ブロックの接続や組み合わせを検討する程度にとどまっていた。

【０００３】すなわち、この離散系シミュレータ装置は、システム全体を見通した制御の構造化／設計といった開発上流工程での利用には援用されなかったし、また、シミュレーションモデルが実機制御用プログラムとは別々の概念で構築されているため、互いに互換性がなかった。

【０００４】

【発明が解決しようとする課題】しかしながら、このような従来の離散系シミュレータ装置では、実機システムのソフトウエアを開発するに際し、システム全体を見通す必要がある開発上流工程（例えば、構造化／設計の工程）において直接利用できないので、この実機システムを良く理解した人間が時間をかけ、目的とするシステムを分析し設計しなければならず、多大な時間を要するという問題点があった。

【０００５】また、シミュレーションモデルが実機システム制御用プログラムとは別々の概念で構築され、互いに互換性がないので、実機システムのプログラムを流用することができず、ソフトウエアの生産性が悪いという問題点があった。

【０００６】特に、近年、実機システムは、ますます大規模化しており、このため所定時間内にシステムを構築するためには、多くの人手をかけなければならいようになってきた。

【０００７】そこで、本発明は、上述の問題点に鑑み、実機システムとの間で共有できる概念に従ってシステムの構造化を行うことにより、仮想環境上でのシミュレータにより作成したモデルの制御構造を実機システム制御の上流設計ツールにおいて利用するプログラム作成支援装置およびプログラム作成支援方法を提供することを目的とする。

【０００８】

【課題を解決するための手段】前述した目的を達成するため、請求項１記載の発明は、シミュレータからシミュレーションモデルを受け、実システムを稼働させる実行コードを作成するプログラム作成支援装置であって、上記シミュレーションモデルから、上記実システム中の機器において実行すべきデータ処理内容を自身内のデータ上に処理プログラムとして有しその実行を管理するプロセスと、このプロセス間をＷｏｒｋが遷移していく順序を指定するＷｏｒｋ経路とからなるコントロールフローを記述したコントロールフロー記述プログラムを作成するコントロールフロー記述プログラム作成手段と、このコントロールフロー記述プログラム作成手段で記述されたコントロールフロー記述プログラムに、上記プロセスが処理する処理プログラムを設定して上記実システムを稼働させる実行コードを作成する実行コード作成手段とを具備することを特徴とする。

【０００９】請求項２記載の発明は、請求項１記載の発明において、上記コントロールフロー記述プログラム作成手段が、上記シミュレータからのシミュレーションモデルから、上記実システム中の機器において実行すべきデータ処理内容を自身内のデータ上に処理プログラムとして有しその実行を管理するプロセスを設定するプロセス設定手段と、このプロセス設定手段で設定されたプロセス間をＷｏｒｋが遷移していく順序を指定するＷｏｒｋ経路を設定するＷｏｒｋ経路設定手段とを有することを特徴とする。

【００１０】請求項３記載の発明は、請求項１記載の発明において、上記実行コード作成手段が、上記コントロールフロー記述プログラムに、ライブラリに格納された上記プロセスが処理する処理プログラムをインクルードして上記実システムを稼働させる実システム稼働プログラムを作成し、これをコンパイルし、さらにリンクして上記実行コードを作成することを特徴とする。

【００１１】請求項４記載の発明は、請求項１記載の発明において、上記実行コード作成手段が、上記コントロールフロー記述プログラムに、エディタで記述された上

記プロセスが処理する処理プログラムをインクルードして上記実システムを稼働させる実システム稼働プログラムを作成し、これをコンパイルし、さらにリンクして上記実行コードを作成することを特徴とする。

【００１２】請求項５記載の発明は、請求項１記載の発明において、上記実行コード作成手段が、上記コントロールフロー記述プログラムをコンパイルし、これに上記プロセスが処理する処理プログラムをリンクして上記実行コードを作成することを特徴とする。

【００１３】請求項６記載の発明は、請求項２記載の発明において、上記プロセス設定手段が、上記シミュレータと上記実システム間で共有する論理的工程をプロセスとすることを特徴とする。

【００１４】請求項７記載の発明は、請求項２記載の発明において、上記プロセス設定手段が、オブジェト指向言語を使用して上記プロセスをオブジェトとして形成することを特徴とする。

【００１５】請求項８記載の発明は、請求項２記載の発明において、Ｗｏｒｋ経路設定手段が、上記オブジェトのプロセス機能を実行可能なプログラム言語で記述する際に、中間言語を介してパラメータを入力するパラメータ入力手段を有することを特徴とする。

【００１６】請求項９記載の発明は、請求項８記載の発明において、上記中間言語が、上記オブジェト指向言語で記述されることを特徴とする。

【００１７】請求項１０記載の発明は、シミュレータからシミュレーションモデルを受け、実システムを稼働させる実行コードを作成するプログラム作成支援方法であって、上記シミュレーションモデルから、上記実システム中の機器において実行すべきデータ処理内容を自身内のデータ上に処理プログラムとして有しその実行を管理するプロセスと、このプロセス間をＷｏｒｋが遷移していく順序を指定するＷｏｒｋ経路とからなるコントロールフローを記述したコントロールフロー記述プログラムを作成し、記述されたコントロールフロー記述プログラムに、上記プロセスが処理する処理プログラムを設定して上記実システムを稼働させる実行コードを作成することを特徴とする。

【００１８】請求項１１記載の発明は、請求項１０記載の発明において、上記シミュレータからのシミュレーションモデルから、上記実システム中の機器において実行すべきデータ処理内容を自身内のデータ上に処理プログラムとして有しその実行を管理するプロセスを設定し、このプロセス設定手段で設定されたプロセス間をＷｏｒｋが遷移していく順序を指定するＷｏｒｋ経路を設定することを特徴とする。

【００１９】請求項１２記載の発明は、請求項１０記載の発明において、上記コントロールフロー記述プログラムに、ライブラリに格納された上記プロセスが処理する処理プログラムをインクルードして上記実システムを稼働させる実システム稼働プログラムを作成し、これをコンパイルし、さらにリンクして上記実行コードを作成することを特徴とする。

【００２０】請求項１３記載の発明は、請求項１０記載の発明において、上記コントロールフロー記述プログラムに、エディタで記述された上記プロセスが処理する処理プログラムをインクルードして上記実システムを稼働させる実システム稼働プログラムを作成し、これをコンパイルし、さらにリンクして上記実行コードを作成することを特徴とする。

【００２１】請求項１４記載の発明は、請求項１０記載の発明において、上記コントロールフロー記述プログラムをコンパイルし、これに上記プロセスが処理する処理プログラムをリンクして上記実行コードを作成することを特徴とする。

【００２２】請求項１５記載の発明は、請求項１１記載の発明において、上記シミュレータと上記実システム間で共有する論理的工程をプロセスとすることを特徴とする。

【００２３】請求項１６記載の発明は、請求項１１記載の発明において、オブジェト指向言語を使用して上記プロセスをオブジェトとして形成することを特徴とする。

【００２４】請求項１７記載の発明は、請求項１１記載の発明において、上記オブジェトのプロセス機能を実行可能なプログラム言語で記述する際に、中間言語を介してパラメータを入力するパラメータ入力手段を有することを特徴とする。

【００２５】請求項１８記載の発明は、請求項１７記載の発明において、上記中間言語が、上記オブジェト指向言語で記述されることを特徴とする。

【００２６】本発明によれば、シミュレータからのシミュレーションモデルから、プロセスと、Ｗｏｒｋ経路とからなるコントロールフロー記述プログラムを作成する。そして、この作成されたプログラムに、プロセスが処理すべきデータ処理内容を記述した処理プログラムを設定して実システムを稼働させる実行コードを作成する。

【００２７】

【発明の実施の形態】以下、本発明に係るプログラム作成支援装置およびプログラム作成支援方法の実施形態を図面に基づいて説明する。

【００２８】図１はプログラム作成支援装置の一実施形態の構成を示すブロック図である。この実施形態のプログラム作成支援装置１は、シミュレータ２から得たシミュレーションモデルに基づいて後述するプロセスおよびＷｏｒｋ経路とからなるコントロールフローを記述したプログラム（以下、コントロールフロー記述プログラムという）を作成するコントロールフロー記述プログラム作成部１１と、後述するようにして実システムを稼働させるテキストでなる実システム稼働プログラムを作成す

る実システム稼働プログラム作成部１２と、実システム稼働プログラムをコンパイルしてこのオブジェクトモジュールを作成するコンパイラ１３と、実システム稼働プログラムのオブジェクトモジュールをリンクして実システムを稼働させる実コードを作成するリンカ１３とから構成されている。

【００２９】ここで、プロセスとは実システム中の機器において実行すべきデータ処理内容を自身内のデータ上に処理プログラムとして有しその実行を管理するものをいい、Ｗｏｒｋ経路とはプロセス間をＷｏｒｋが遷移していく順序を指定する経路をいう。

【００３０】実システム稼働プログラム作成部１２が、実システム稼働プログラムを作成する内容を詳細に説明する。

【００３１】実システム稼働プログラム作成部１２は、コントロールフロー記述プログラムに、処理プログラムインクルードライブラリ１６に格納されている各プロセスが処理する処理プログラムをインクルードし、上記実システム稼働プログラムを作成するか、または、コントロールフロー記述プログラムに、エディタ１５で記述された各プロセスが処理する処理プログラムをインクルードして実システム稼働プログラムを作成する。

【００３２】次に、本発明に係るプログラム作成支援装置は、「プロセス」という概念を基本にして制御モデルを表現するので、以下、その内容を説明する。

【００３３】説明を簡単にするため、図２に示すようなシステム構成の場合を例にとり説明する。

【００３４】このシステムは、機器Ａと，機器Ｂと，ｂｕｆｆｅｒ＿Ｘと、ｂｕｆｆｅｒ＿Ｙとで構成されており、機器AにおいてＷｏｒｋに対するある所定の処理１を行い、次に、機器ＢにおいてＷｏｒｋに対するある所定の処理２を行った後、Ｗｏｒｋ＿Ｘ，Ｙをそれぞれｂｕｆｆｅｒ＿Ｘ，Ｙに振り分ける作業を行っているものとする。

【００３５】このシステムに「プロセス」の概念を割り当てると、図３に示すように、「プロセス」というオブジェト（図中ではｐｒｏｃｅｓｓと表記されている）間を、Ｗｏｒｋ転送の流れ（以下、Ｗｏｒｋ経路といい、図中ではＷｏｒｋ＿＊と表記されている）で結んだモデルに表現することができる。

【００３６】すなわち、図２に示したシステムは、機器Ａがｐｒｏｃｅｓｓ＿１、機器Ｂがｐｒｏｃｅｓｓ＿２およびｐｒｏｃｅｓｓ＿３に表現され、また、ｐｒｏｃｅｓｓ＿１に流入するＷＲ＿１，ｐｒｏｃｅｓｓ＿１から流出してｐｒｏｃｅｓｓ＿２に流入するＷＲ＿２，ｐｒｏｃｅｓｓ＿２から流出してｐｒｏｃｅｓｓ＿３に流入するＷＲ＿３，ｐｒｏｃｅｓｓ＿３から流出してｂｕｆｆｅｒ＿Ｘに流入するＷＲ＿４および，ｐｒｏｃｅｓｓ＿３から流出してｂｕｆｆｅｒ＿Ｙに流入するＷＲ＿５とするモデルで表現することができる。

【００３７】なお、このモデルを構成するオブジェトは、論理的な内容を区切りとして設定されているので、上述した機器Ｂについてのように、機器の数と一致している必要はない。

【００３８】また、Ｗｏｒｋ経路は、矢印で示された順序であるが、常にＷｏｒｋの実移動を伴うものとは限らない。例えば、機器Ｂではｐｒｏｃｅｓｓ＿２の終了後、Ｗｏｒｋを同じ位置に固定したままの状態でｐｒｏｃｅｓｓ＿３が起動される。この場合には、Ｗｏｒｋの実移動はないが、論理上ではＷｏｒｋがｐｒｏｃｅｓｓ間を遷移するということで（図中のＷＲ＿３）定義される。

【００３９】ここで、従来より使用されていたペトリネット表現を用いてプロセスの具体例を説明する。

【００４０】上記図２中のシステム中における機器Ａの動作をペトリネットで表現すると、図４に示すようになる。

【００４１】すなわち、『機器Ａの稼働状態』および『Ｗｏｒｋに対する処理の進行状態』がマーキングで表現されることになる。しかし、この表現方法では、Ｗｏｒｋに対する機器の実行すべき処理内容を記述する部分が存在しない。従って、実システムで利用することができる制御構造を記述するためには、少なくとも処理の内容を記す枠組みが必要となる。

【００４２】そこで、図５に示すように、工程の『処理内容』を「処理プログラム」と定義して追加する。

【００４３】つまり、『機器Ａでの作業』を一般化して『工程』と表現しているのである。なお、図中の「処理プログラム」がプレースＰ＿３で実行される作業内容である。ここで、作業内容としては、例えば、Ｗｏｒｋを切る、曲げる、穴を開けるなど必要な処理が表現されることになる。

【００４４】さらに、図５を、機器Ａから『ＰＵＬＬ型機器に対する出力』も考慮して一般化すると、図６に示すようになる。

【００４５】つまり、①機器Ａの作業を『工程』とし、②『Ｗｏｒｋ：処理待ち』プレースを汎用的起動要素としての「処理待ち」とするような一般化を行い、さらに、『ＰＵＬＬ型機器』への出力に対応のため、プレース「出力」，「出力要求」およびゲート機構を有するトランジッション「Ｔ＿３」を追加した。

【００４６】このようなモデル構成において、特にＷｏｒｋ転送機能および「処理プログラム」の実行管理機能の部分をメソッドとしてもつオブジェクトを、クラス「ｐｒｏｃｅｓｓ］のインスタンス・オブジェクトと定義することができる。

【００４７】上記の点を考慮に入れて、次に、図３のシステムをオブジェクト指向言語「Ｃ＋＋」を用いて表現してみると図７および図８に示した記述になる。

【００４８】この例において、オブジェクト「ｐｒｏｃ

ｅｓｓ］は、図３に示したペトリネットの内容どうりに動作するよう記述されている。

【００４９】なお、機器Ｂの「ｐｒｏｃｅｓｓ　Ｙ」は、「ＰＵＬＬ型」入力であり、また、「ｐｒｏｃｅｓｓ　Ｚ」は、Ｗｏｒｋの物理的移動の伴わない「ＰＵＳＨ型」とした。

【００５０】以下、このプログラムの各部を説明する。

【００５１】

【演算子定義と中間言語表記】について

中間言語表記とは、「ｐｒｏｃｅｓｓ　型」オブジェクトへのメッセージ通信を、「ｐｒｏｃｅｓｓ　型」に対して再定義された演算子（図７中の符号７１部分参照）を用いて記述するものである。

【００５２】このような中間言語表記においては、「ｐｒｏｃｅｓｓ　型」のＸ，Ｙおよび「ＶＥＣＴ　型」のＺに対して以下のような演算が可能となる。

【００５３】すなわち、処理を実行する場合においては、Ｘ＋Ｚ：Ｘが処理を実行した後にＷｏｒｋを保持し（出力要求待ち状態）、Ｘ＞Ｙ：ＸからＹの順に処理を続けて実行するようになる。また、Ｗｏｒｋを転送する場合においては、Ｘ＜＜Ｙ：ＸがＹにＷｏｒｋの転送を要求（ＰＵＬＬ）し、Ｘ＞＞：ＸＧＡＷｏｒｋを強制転送（ＰＵＣＨ）するようになる。

【００５４】このような、中間言語表記を用いることにより、図８に示した機器制御に関する記述が、より簡素な表現となっていることは、図９（中間言語表記を用いなかった場合の機器制御に関する記述部分）との比較により明らかである。

【００５５】

【共通定義】について

ラベル定義では「ＶＥＣＴ　型」に相当する定数値を設定している。また、「ＶＥＣＴ　型」変数ＩＯ＿Ａ，ＩＯ＿Ｂは、『センサ』などにおいて物理的なＷｏｒｋ入力を検出した際、０ｘ００　→　０ｘ０１と変化するものと定義する。さらに、このプログラムにおいては、説明を簡単するため、その値が読み出されるまでラッチされているものとする。

【００５６】

【機器Ａの制御】について

機器Ａにおいて、物理的なＷｏｒｋの入力は、ＩＯ＿Ａをとうして通知され、「ｐｒｏｃｅｓｓ　Ｘ」は、『Ｗｏｒｋ：処理待』状態となったことを認識する（Ｘ＋ＩＯ＿Ａ）。すると、「ｐｒｏｃｅｓｓ　Ｘ」は、『工程：ＩＤＬＥ』状態となり次第、プログラム実行を開始する。また、後続が「ＰＵＬＬ型」なので、出力に関する制御は不要となる。

【００５７】

【機器Ｂの制御】について

機器Ｂにおいて、「ｐｒｏｃｅｓｓ　Ｙ」は、『工程：ＩＤＬＥ』状態になると、「ｐｒｏｃｅｓｓ　Ｘ」にＷｏｒｋを出力させる（Ｙ＜＜Ｘ）。次に物理的なＷｏｒｋ入力は、ＩＯ＿Ｂを通して通知され、「ｐｒｏｃｅｓｓ　Ｙ」は、『Ｗｏｒｋ：処理待』状態になったことを認識する（Ｙ＋ＩＯ＿Ｂ）。

【００５８】次に、「ｐｒｏｃｅｓｓ　Ｚ」は、「ｐｒｏｃｅｓｓ　Ｙ」から処理終了を通知され、『Ｗｏｒｋ：処理待ち』状態となったことを認識する（Ｙ（＋ＩＯ＿Ｂ＞　Ｚ）。

【００５９】このことにより、「ｐｒｏｃｅｓｓ　Ｚ」は、物理的な移動を伴わずにＷｏｒｋを受けとり、「ｐｒｏｃｅｓｓ　Ｙ」に連続して処理を開始できる。

【００６０】続いて、離散系シミュレータ「ＱＵＥＳＴ」から「プロセス」により構築されたモデルへのダウンロードについて説明する。

【００６１】図１０は図２の制御対象を「ＱＵＥＳＴ」によりダウン・ロードする際の説明を示すモデル構築図である。

【００６２】ここで、制御対象を図２に示した機器Ａおよび機器Ｂとし、かつ、その機器Ａをバッファ、機器Ｂをワーク・セルとするものとする。

【００６３】図に示すように、機器Ａは、キュー型のバッファであり、機器Ｂは、１つの入力、２つの出力を有する「ＰＵＬＬ型」のワーク・セル（加工機）であるとする。

【００６４】そして、ワークセルの動作ロジックは、上述の条件に基づき、機器Ａについてはキュー処理（先入れ先出し方式）を行うＦＩＦＯ、機器Ｂについて加工に関するＳｐｅｃｉｆｙ、また、振分け作業に関するものをＦｉｘｅｄとして設定する。このＳｐｅｃｉｆｙが、加工の実行を行い、このＦｉｘｅｄは、２つの出力とＷｏｒｋ種別との振り分けを設定している。

【００６５】次に、「中間言語」の仕様を用いた記述を図１１に示す。これは、図８と同様なものであるが、「ＱＵＥＳＴ」の表現と対応したラベル名を使用している。

【００６６】また、「ＶＥＣＴ　型」変数ＩＯ＿Ａについては、「ｐｒｏｃｅｓｓ　Ｑｉｎｇ」においてバッファ機能を実現するため、処理が任意に実行可能となるよう、物理的なＷｏｒｋ入力を検出した際、０ｘ００　→　０ｘ０３と変化するものと定義しておく。

【００６７】このように、「ＱＵＥＳＴ」上でのモデル構築で行った各種の設定情報から図１１の記述を導くことは、対応関係が明確なので容易である。しかし、図１１の記述だけでは、制御内容が不十分であり、「ＱＵＥＳＴ」においてはメニュー選択で設定した「ＦＩＦＯ」などのＷｏｒｋに対応する具体的処理が実行できない。

【００６８】従って、これらの部分を次のように各「処理カリキュラム」に割り当て、その制御を記述しておく必要がある。

【００６９】

すなわち、

```
Void Fifo(){
        ＦＩＦＯ動作を実行するためのハードに依存した制御
}
Void Spfy(){
        Ｗｏｒｋ加工動作を実行するためのハードに依存した制御
}
Void Fixdy(){
        Ｗｏｒｋ出力振分け動作を実行するためのハードに依存した制御
        ただし、出力タイミング制御のみは、オブジェクト「Ｒｏｕｔ」内
        で行なう
}
```

【００７０】上述の制御は、採用する機器に依存した制御プログラムとなるので、その内容を状況に応じて記述する必要があるが、「ＦＩＬＯ」などの代表的な動作についてはライブリとしておけばよい。

【００７１】そして、最後に、これらの「中間言語表記」による記述を「Ｃ＋＋コンパイラ」で処理するれば、実システムの機器で実行可能なコードが生成される。

【００７２】つまり、離散系シミュレータから実システムに、ダウンロードが完了することになる。

【００７３】上述した実施形態のプログラム作成支援装置では、実システム稼働プログラム作成部１２で、エディタ１５または処理プログラムインクルードライブラリ１６で作成された各プロセスの処理プログラムを、コントロールフロー記述プログラムにインクルードし、これをコンパイラ１３でコンパイルし、さらにリンカ１４でリンクするような構成になっているが、それ以外として、図１２に示すような構成をなしてもよい。

【００７４】すなわち、図１２に示すプログラム作成支援装置は、シミュレータ２から得たコントロールフロー記述プログラムを作成するコントロールフロー記述プログラム作成部１１と、コントロールフロー記述プログラムをコンパイルしてこのオブジェクトモジュールを作成するコンパイラ１３と、処理プログラムリンクライブラリ１７に格納された各プロセスが処理する処理プログラムを、コンパイラ１３において作成されたオブジェクトモジュールにリンクして実システムを稼働させる実行コードを作成するリンカ１４とで構成してもよい。

【００７５】

【発明の効果】上述の本発明によれば、シミュレータからのシミュレーションモデルから、プロセスおよびＷｏｒｋ経路が設定されるため、各種離散系シミュレータ間での「制御構造」の共有化および実システムへのダウン・ロードが可能となり、離散系シミュレータの「制御構造の上流設計ツール」として活用および実システムのプログラムの自動生成が推進できる。

【００７６】このため、上流工程のシステム分析・設計を容易にし、また、プログラムの生産性を向上させることができる。

【図面の簡単な説明】

【図１】本実施例のプログラム作成支援装置の一実施形態の構成を示すブロック図。

【図２】実システムの一例を構成を示すブロック図。

【図３】図２に示した構成を「プロセス」を適用したモデルに示したブロック図。

【図４】図２中の機器Ａをペトリネット表現で示した図。

【図５】図４に工程処理を含ませた図。

【図６】図２中の機器Ａを工程処理および「ＰＵＬＬ型機器への出力」を考慮して一般化した図。

【図７】図３のシステムをオブジェト指向言語「Ｃ＋＋」で記述した図。

【図８】図３のシステムをオブジェト指向言語「Ｃ＋＋」で記述した図。

【図９】中間言語表記を含まない場合の機器制御を記述した図。

【図１０】図２の制御対象を「ＱＵＥＳＴ」によりダウン・ロードする際の説明を示すモデル構築図。

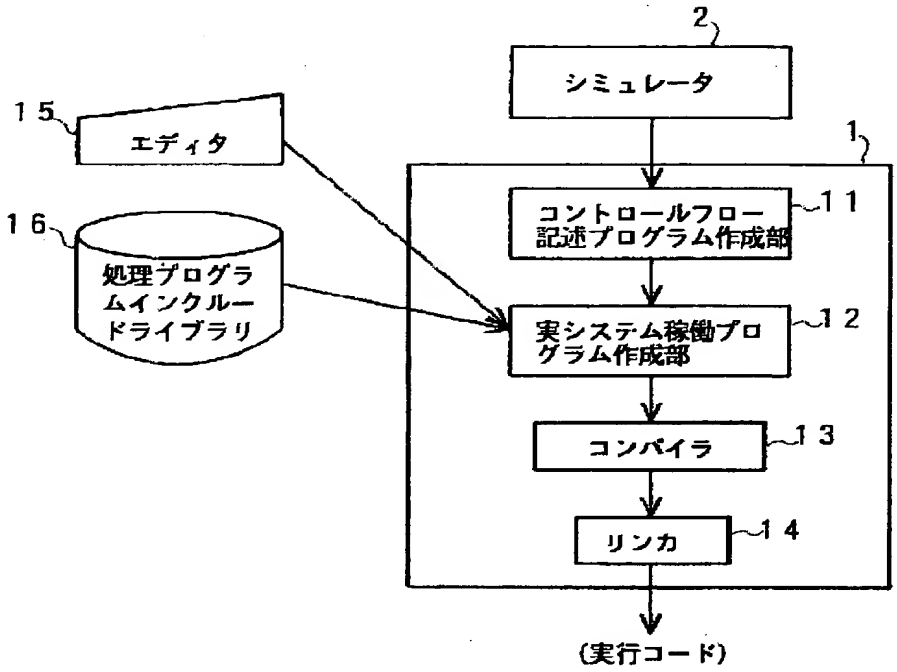
【図１１】システムを記述したプログラムのうちの「中間言語表記」を示す図。

【図１２】他の実施例のプログラム作成支援装置の一実施形態の構成を示すブロック図。

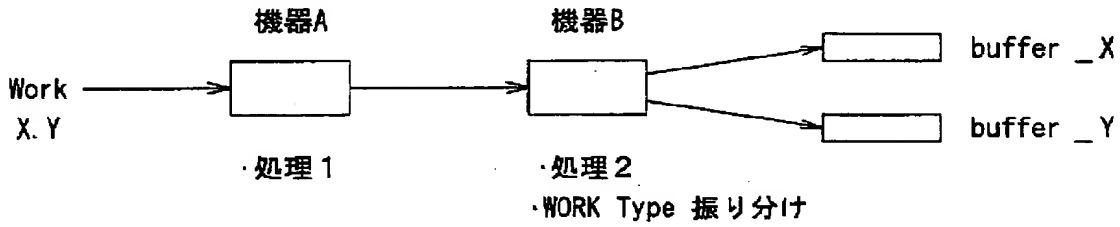
【符号の説明】

１	プログラム作成支援装置
２	シミュレータ
１１	コントロールフロー記述プログラム作成
１２	実システム稼働プログラム作成部
１３	コンパイラ
１４	リンカ
１５	エディタ
１６	処理プログラムインクルードライブラリ
１７	処理プログラムリンクライブラリ

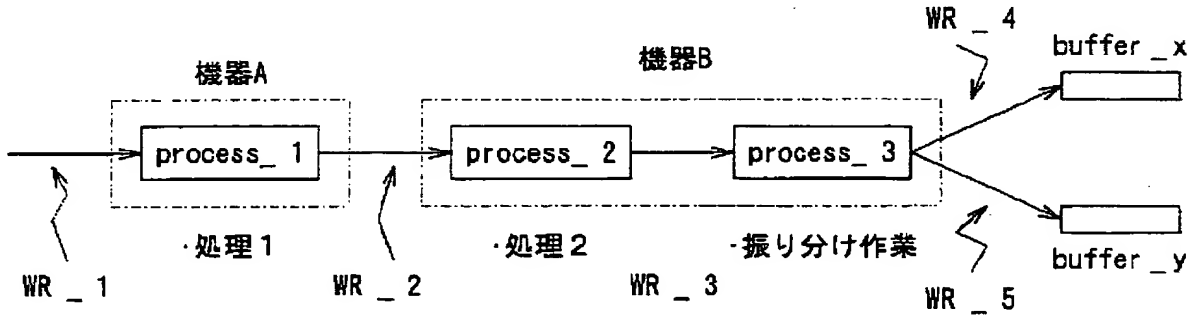
【図1】


2
シミュレータ
15
エディタ
1
16
処理プログラムインクルードライブラリ
コントロールフロー記述プログラム作成部
11
実システム稼働プログラム作成部
12
コンパイラ
13
リンカ
14
(実行コード)

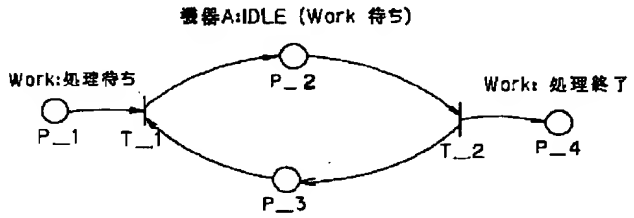
【図2】


機器A
機器B
Work
X. Y
buffer _X
buffer _Y
・処理1
・処理2
・WORK Type 振り分け

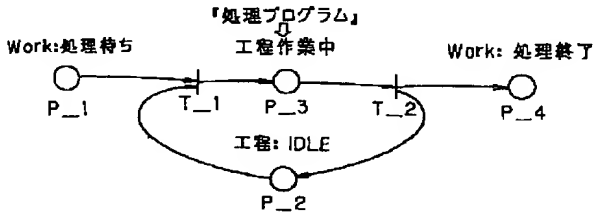
【図3】


WR _ 4
buffer _x
機器A
機器B
process_ 1
process_ 2
process_ 3
buffer _y
・処理1
・処理2
・振り分け作業
WR _ 1
WR _ 2
WR _ 3
WR _ 5

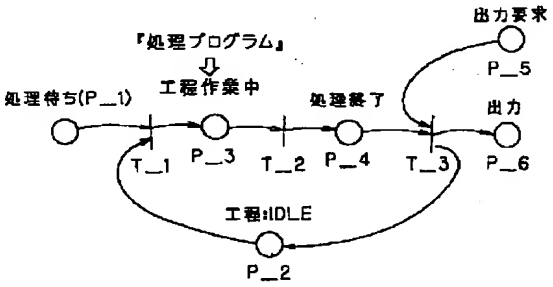
【図4】



【図5】



【図6】



【図8】

【機器制御プログラム】

```
VECT IO_A, IO_B                    // 物理的IO信号(ｴｯｼﾞ)の入力用

viod main_A()                      // 機器A用main 関数
 {
  process X(Code_X);               // クラス宣言
  while(1)  X + IO_A;              // Xを実行
}
viod main_B()                      // 機器B用main 関数
{
  process Y(Code_Y);               // クラス宣言
  process Z(Code_Z);               // クラス宣言

  while(1) {
    Y << X;                        // Xから Work を PULL
    Y + IO_B ; Z;                  // Y, Z の順に実行
    Z >> ;                         // Z PUSH 出力
  }
}
```

【図7】

```
//【共通定義】

    #define VECT  int
    #define P_1  0x01                    // 『WORK入力』
    #define P_2  0x02                    // 『工程IDLE』
    #define P_4  0x08                    // 『処理終了』
    #define P_5  0x10                    // 「出力要求』

//【クラス定義】

    class process  {
        int   (*Func)        「処理プログラム」を記述する関数へのポインタ
        VECT  MARKing ;      「process」の状態変数
      public :
        process(int (*pFune)()) Func = pFunc ;          //object  構築関数
       ~process() ;                                     //object  消滅関数
        operator + (VECT);    // 『 + 』演算の定義
        operator *(VECT);     // 『 *』演算の定義
        operator >> (VECT);   // 『>> 』演算の定義
        operator << (VECT).   // 『<< 』演算の定義
        operator  | (VECT);   // 『 | 』演算の定義
    };

    process : :operator+(VECT inMK) {          // 【『 * 』演算の定義】
      MARKing  |=  inMK ;                      // Token の追加設定
      if (MARKing & 0x03 ==  0x03) {           // Token  : (P_1 & P_2) ?
        MARKing = (MARKing & 0xF0)+ 0x04;      // Token - P_3
        (*Func) ();                            // Func の参照する関数を実行
        MARKing  =  (MARKing & 0xF0)+ 0x08;    // Token -> P_4
      }
      if (MARKing & 0x18  ==  0x18) {          // Token:  (P_4 & P_5) ?
        Out Work();                            // 物理的  WORK 出力(外部関数)
        MARKing  =  (MARKing & 0xE0)+ 0x02;    // Token -) P_2
      }
    }

    VECT process ::operator*(VECT  inMK)  {    // 【『 * 』演算の定義】
      inMK &=  MARKing;                        //  入力変数  AND 内部状態
      return(inMK) ;
    }                                    //

    process& process ::operator >> ()  {       // 【『 >>』演算の定義】
      this+P_5;                                // PUSH 出力
    }

    process& process ::operator << (process  Y) {   // 【『 << 』演算の定義】
      if(this*P_2) Y+P_5;                           //  PULL 出力
    }

    process& process ::operator  | (process   Y) {  //【『  |  』演算の定義】
        if(this*P_4) {                             //
            this+P_5;                              // this:  Token -> P_2
           Y+P_1;                                  //    Y:  Token -> P_1
        }
    }
```

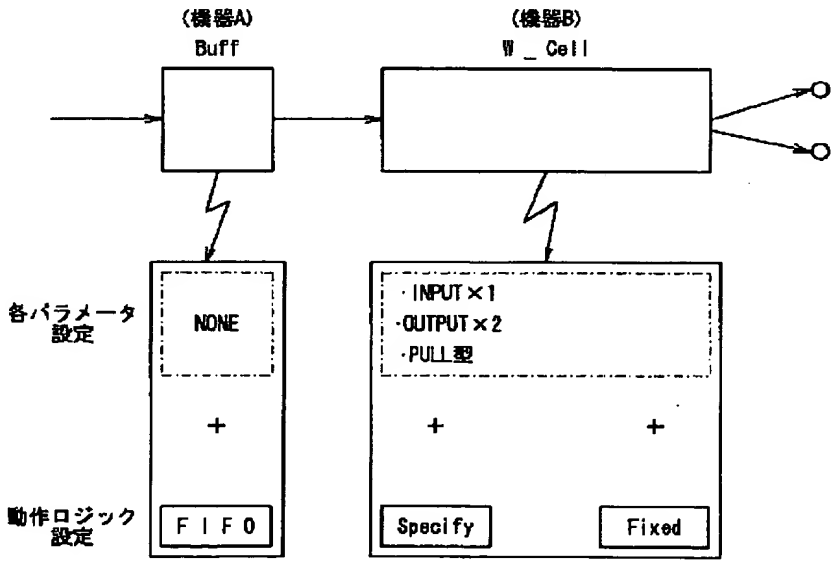
71

【図９】

【機器制御プログラム】

```
viod main_A()                          // 機器A用  main 関数
{
 process        Y(Code_A);             // クラス宣言

 while(1) {
   if (IO_A)  X+P_1;                   // Work の物理的入力検知      X: 処理開始
 }
}

void main_B ()                         // 機器B用  main 関数
 {
  process       Y(Code_A),             // クラス宣言
  process       Z(Code_B);             // クラス宣言
  VECT          Link  = 0              //  「処理終了」通知用

  while(1) {
    if  (Y*P_2)  X+P_5;                // Y: IDLE ならば  X: PULL 入力
    if  (IO_B) {                       // Work の物理的入力検知
      Y+P_1;                           //  Y: 処理開始
      if (Y*P_4)  {                    //  Y: 処理終了  ?
        LINK = 0x01;                   //  フラグON    (Token  -) P 1)
        Y+P_5;                         //  Y: PUSH出力
      }
    }

    Z+Link ;                           // Z:処理開始  (Token -)  P 1)
    Link = 0x00;                       //  フラグOFF
    Z+P_5 ;                            //  Z:  PUSH出力
  }
}
```

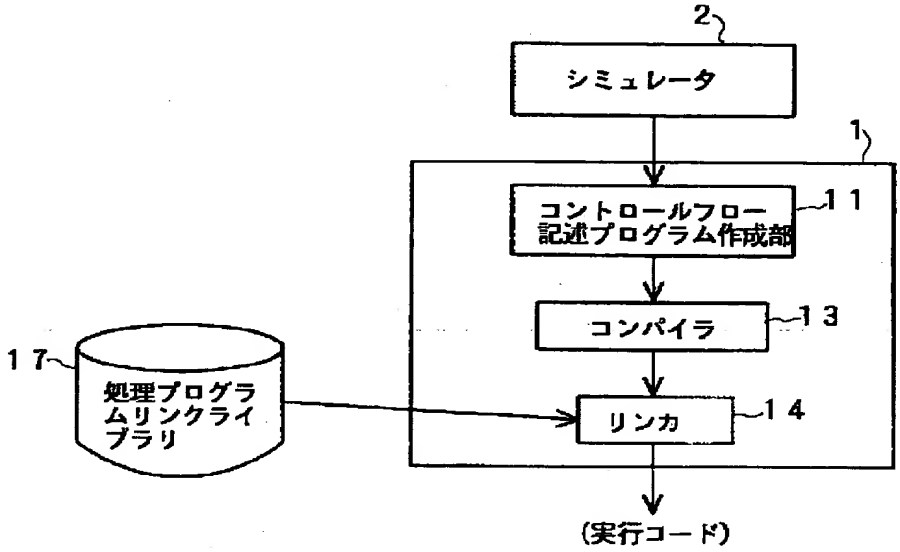
【図１０】



【図11】

```
//『中間言語による記序述』

    int      B_in, W_in;                          // Buf, Wcell の入力検知IO


//Buf の制御

     vod Buf main()
     {
      process Qing(Fifo) ;                        //クラス宣言
      while(1) Qing + IO_A;                       // Qing を実行
     }


//W_Cell  の制御

    void Wcell main()
     {
      process proc(Spfy);                         //クラス宣言
      process Rout(Fixd);                         //クラス宣言

      while(1) {
        Proc (( Qing ;                            // Buf から  Work をPULL
        Proc + IO_B | Rout;                       // Proc. Rout の順に実行
        Rout ));                                  // Rout PUSH 出力
      }
    }
```

【図１２】


2
シミュレータ
1
コントロールフロー
記述プログラム作成部
11
コンパイラ
13
17
処理プログラ
ムリンクライ
ブラリ
リンカ
14
(実行コード)